CSE 568: Robotics Algorithms
(Fall 2018)
Instructor: Karthik Dantu

Project 3: Colorizing the Prokudin-Gorskii photo collection

October 24, 2018

Report By:
Siddheswar Chandrasekhar

Objective

The objective is to implement some basic image processing algorithms to detect features to frame alignment.

Task 1: Simple Color Images

We are given six images. Each image is a concatenation of three separate glass plate images, one for each color channel (R, G and B). The task is to take each of the six images, align the three plate images as three color channel images and save the resultant color image.

We observe that each image is of size 1024x400. Since each image is a concatenation of three image plates, we deduce that each plate is of size 341x400 (approx.). We hence find the individual plates, which are also the three color channels of the image using the pseudo-code:

```
b = img(1:floor(x/3)+1 , 1:y);
g = img(floor(x/3):2*floor(x/3) , 1:y);
r = img(2*floor(x/3):x-1 , 1:y);
```

We now have our three color channels and we place each channel on top of each other using the *cat()* function of Octave as:

```
colorimg = cat(3 , r , g , b);
```

Our original image looks as follows:

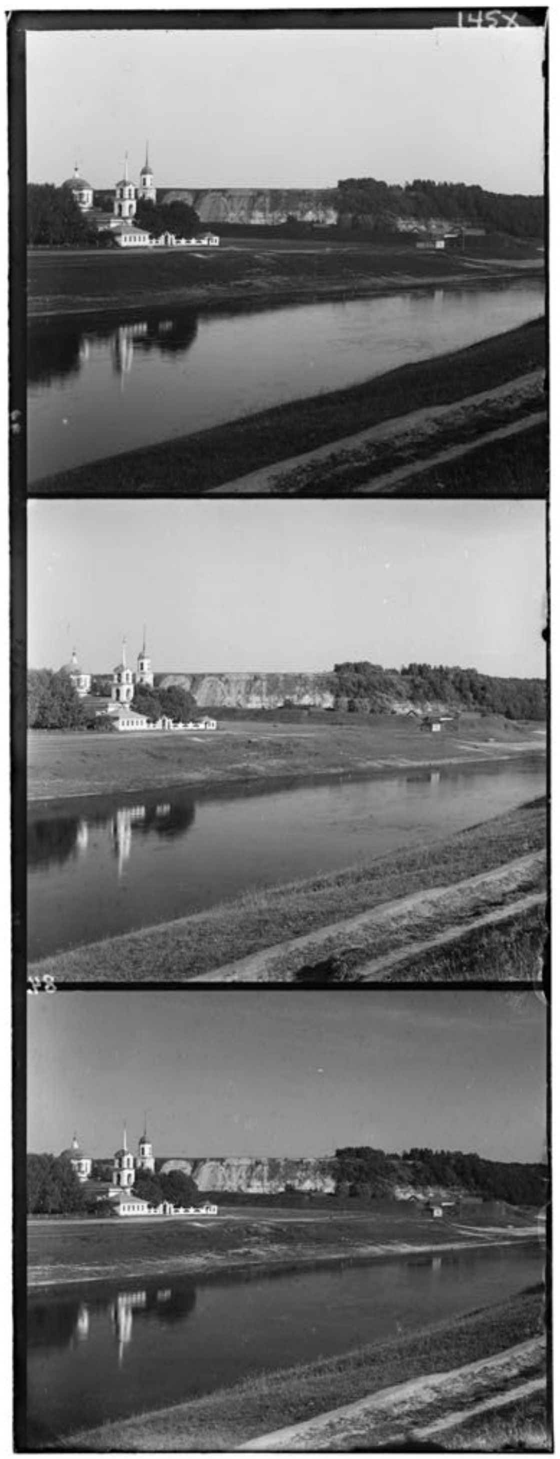

Fig 1.1 Original Image

Our colored image looks as follows:

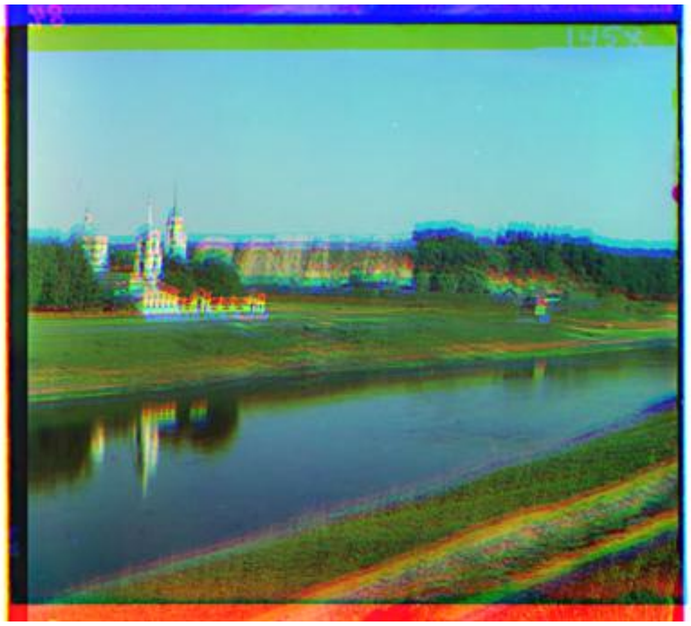
Fig 1.2 Colored Image

As expected, the image looks blurred. This is because they are not aligned correctly.

Task 2: Aligning Images

As seen above, just overlapping the individual channel images creates blurred images. In this step, we will find the correct alignment.
The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-15,15] pixels), score each one using some image matching metric, and take the displacement with the best score. There is a number of possible metrics that one could use to score how well the images match. The simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply sum(sum((image1-image2)2)). Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: (image1./|image1| and image2./|image2|).

We first perform SSD alignment.

As we know, Sum of Squared Differences (SSD) or Mean Squared Error (MSE) is given by:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

The best score is the one with the least SSD value.

In our case, we notice that the border pixels mess with the SSD calculations. We hence drop the border pixels from the image using the pseudo-code:

```
new_image_channel =

image_channel (offset : <height_of_image_channel> - offset , offset : <width_of_image_channel> - offset)
```

Here, offset is the number of number of rows / columns we decide to drop from the image channel to get rid of the borders.
In our program, we choose an offset of 30 as we find that 30 perfectly gets rid of the borders on all sides.

We then take a window size of [-15,15] and search for a best score by running the R and G channels over the B plane.

After finding the best scores for the R and G channels individually, we calculate the new offsets for R and G channels using these scores and then overlap them with the B channel.

We then finally get a clean image as follows:

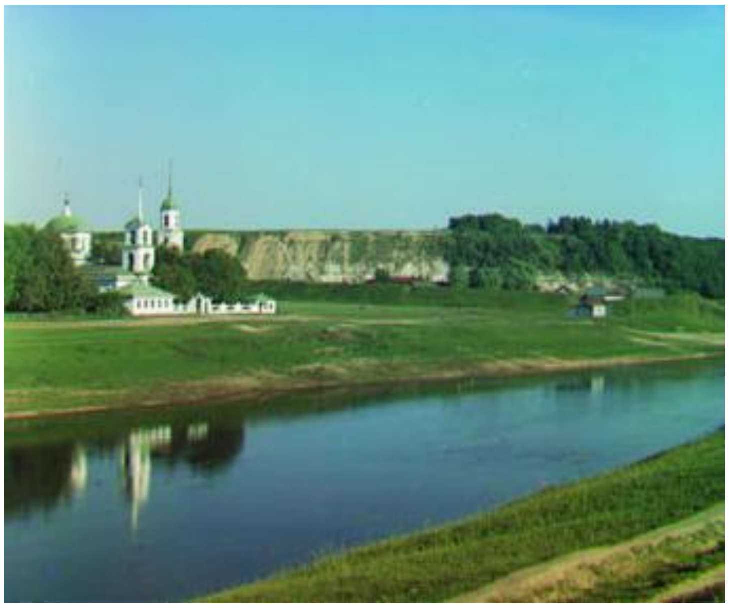
Fig 2.1 SSD-aligned image

We now perform NCC alignment.

As we know, Normalized Cross-Correlation (NCC) is given by:

$$\frac{1}{n}\sum_{x,y}\frac{1}{\sigma_f\sigma_t}\left(f(x,y)-\overline{f}\right)\left(t(x,y)-\bar{t}\right).$$

The best score is the one with the maximum NCC value.

For this task, we use the *normxcorr2()* function from the *image* package in Octave.

The *normxcorr2()* function takes two parameters, *template* and *image*. The resulting output is a matrix containing the correlation coefficients. [1]

We hence, cut-out a part of our R and G channels to use as template over the B channel (which is the image). The key here is to cut out a part that represents a distinctive feature from the image. By looking at the images, and trying out several other combinations, we finally deduce that the center of the upper half of the image, makes a good template.

We hence set our template to:

```
template = <image_channel> (100:150 , 100:150);
```

We then pass this template along with the B channel as our image to the *normxcorr2()* function.

We do this for both R and G planes and find the best scores for each. We then, like SSD, calculate the new offsets for R and G channels using these scores and then overlap them with the B channel.

We get an NCC-aligned image as follows:

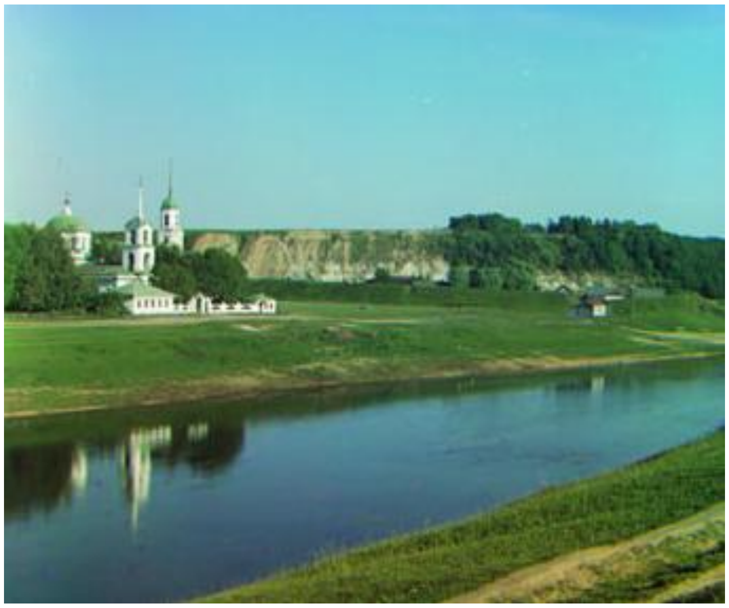
Fig 2.2 NCC-aligned image

This is similar to the SSD-aligned image.

Observations

We find the following alignment shifts:

Image 1 G channel alignment shift using SSD = [-4, -2] R channel alignment shift using SSD = [-9, -1] G channel alignment shift using NCC = [-4, -2] R channel alignment shift using NCC = [-8, -1]	Image 2 G channel alignment shift using SSD = [-3, -2] R channel alignment shift using SSD = [-8, -2] G channel alignment shift using NCC = [-3, -2] R channel alignment shift using NCC = [-8, -2]
Image 3 G channel alignment shift using SSD = [-6, -3] R channel alignment shift using SSD = [-13, -4] G channel alignment shift using NCC = [-5, -2] R channel alignment shift using NCC = [-13, -4]	Image 4 G channel alignment shift using SSD = [-14, 15] R channel alignment shift using SSD = [-12, -1] G channel alignment shift using NCC = [-3, -1] R channel alignment shift using NCC = [-12, -2]
Image 5 G channel alignment shift using SSD = [-4, -2] R channel alignment shift using SSD = [-10, -4] G channel alignment shift using NCC = [-3, 1] R channel alignment shift using NCC = [-9, -2]	Image 6 G channel alignment shift using SSD = [1, 0] R channel alignment shift using SSD = [-4, -1] G channel alignment shift using NCC = [3, 1] R channel alignment shift using NCC = [-4, -1]

References

1. normxcorr2
   https://www.mathworks.com/help/images/ref/normxcorr2.html